國科會計畫成果報告：以仲介者為基礎的廣域軟體再利用環境之研究與設計(3/3)

國科會計畫編號：NSC 89-2213-E-009-001

鍾乾癸，劉文謙，翁獻欽
國立交通大學資訊工程研究所
新竹市大學路 1001 號
e-mail:[cgchung, wcliu, scueng]@csie.nctu.edu.tw

摘　要

軟體再利用已是目前公認最能夠提升軟體產能及品質之軟體工程技術，在各項軟體再利用技術中，生產線(Product-Line)的軟體再利用，是以整合由上到下的方式，針對同質的產品有系統的規劃共同的需求，架構與元件等可再利用資源，以提升再利用的效率，因此，生產線的軟體再利用逐漸在軟體工程研究與應用的領域受到重視。然而目前的相關研究多停留在仍停留在概念性的探討，並未與現有完整的軟體開發方法結合，因此我們在本研究中，將結合生產線中最重要的特徵概念[14]與現有最廣為使用的 Rational Unified Process[16]，提出一套完整的生產線(Product-Line)的軟體再利用(包含 for reuse 與 with reuse[2])的方法，藉由特徵模型來描述產品線的特徵，藉由完整的軟體發展程序 RUP 來輔助產品線的開發。

1. 簡介

軟體再利用技術已是目前公認最能夠提升軟體產能及品質之軟體工程技術[1]~[3]。典型的軟體再利用技術是將可被再利用之軟體產物(Software Artifacts)以軟體元件的型式集中於一元件儲存庫(Component Repository)中，當軟體開發者欲開發一新的軟體系統時，透過搜尋機制到軟體元件庫中尋找合用的軟體元件，在了解其功能與行為後，篩選出合適的軟體元件，直接引用或略加修改而組合成為新的軟體系統。因此軟體開發者不需自行開發軟體系統之所有組成單元，不但可以節省開發成本、時間及提升產能，且由於現存之軟體元件均已透過驗證及測試，對整體系統之品質驗證也可節省不少時間。[4][5]

然而這類被動式的再利用方式，再利用者只能從前人累積的元件挑選適合的進行再利用，一來再利用者所需要的的元件並不一定存在，而且其功能也難以完全符合再利用者的需求，因此效果有限。另一類的方式即是針對未來即將開發的各類軟體，建立再利用的計畫，也就是針對這些軟體建立可再利用的軟體需求，軟體架構，以及軟體元件等，以整合由上到下的方式，有效的規劃軟體公司內部可再利用資源，此即所謂生產線(Product-Line)的軟體再利用[3][6][7]。

所謂生產線是指享有共同的特徵以及共同的商業目標的軟體集合。從一個公司的角度來看，生產線泛指一個公司底下同一家族系列的產品，這些產品都擁有彼此間該有的共同產品功能，同時每一個家族的各產品成員間，各有在功能需求上不同的鮮明特徵以分辨彼此成員的差異。其他如各家軟體公司出產一系列的家族產品，或推出加入新功能、及更新版本的軟體元件等，都是屬於同一生產線上發展新產品的策略。而生產線的特點在於，產品本身會有其他產品所沒有(或相似)功能，這些都有可能成為產品的特徵，我們希望由產品特徵來分辨與其他產品的不同。特徵囊括有關系統提供的功能性服務，還有非功能性如發展作業平台的差異、效能、外部硬體支援等，都是用以區分不同產品的特徵。為了因應市場需求的快速變化，使用者隨時可能變更系統功能需要，如何精確辨認並找出合適的軟體元件配合使用或開發愈行重要。在這種情況下，若能直接由產品特徵的選取出適當元件，將會讓軟體發展速度更快。因此，生產線軟體開發已經成為目前軟體工程中重要的研究[8]~[9]。

目前以生產線為理念的軟體發展方法如 RSEB[10][11]、PuLSE[12]、FAST [6]、FORM[13]，仍停留在概念性的探討，而缺乏完整的生產線軟體開發方法。生產線軟體開發中最關鍵的問題在於整個生產線上不同版本產品控管的技術問題。同一系列的產品在不同時期會因應不同的需要，而有功能上、環境上、使用對象等變化要素存在。為了應因不同需求的變化，如何用最少的花費得到最多的成果，即是軟體再利用及生產線上軟體發展最主要的探討議題。同一個生產線上的產品主要會有以下幾種版本的可能：

(1) 作業平台改版: 為了不同客戶的作業環境需求，除了原有的作業平台版本外，還有可能發展其他作業平台版本。

(2) 能力特徵改版: 新的需求通常會不斷出現，一般會因應不同使用者的需求，而提供不一樣的系統功能。也許是使用者需求的改變、功能服務的新增、或將原有的功能強化等，皆會造成不同版本產品的差異。

(3) 程式語言改版: 因不同程式語言的特性差異，為了改善產品整體效益，可能針對產品內部某子系統，如介面呈現、輸出輸入處理等部分，改用別種開發語言改善產品。

(4) 國際語言變化改版: 考慮不同國家的使用者需求，作業平台會有不同語系的版本，為了讓產品能作業在不同語系的作業環境下，生產線應考慮發展不同國際語言版本產品。

然而現有的狀況則是一個軟體公司在產品開發初期不知此產品是否會成功，或者對此領域還不夠充分瞭解所以不敢大規模開發。而當產品被市場接受後，OEM 客戶會因不同的需求而要求修改產品，此時產品線上就有不同版本產品開發的需求，所以新的產品就一直出現，新的開發專案就會產生。這時線所要面臨的問題是，如何快速因應市場的需求，在最短的時間內用最少的花費將新版產品開發出來。面對新的需求不斷改變，軟體工程師不可能從頭分析設計系統，所以應當考慮在產品線上探討在發展程序的過程中是否有可再利用的資產。基於上述所提的變化，皆關係到一個產品不同版本的問題。針對此一問題，本論文旨在於如何應用產品線的觀念於新產品的開發流程上，以便在產品成功後，可以配合其他客戶需求用最少的時間與資源開發新產品。

為了能讓後來開發的新產品能有效再利用，我們在開發初版產品時必須要有一個有效的方法，讓現階段的開發工作成為一個產品線的準備工作，在產品成功後就可以在最短的時間內將新版的產品開發出來。所以在開發初版產品時要有下面的幾個前提：

(1) 要有一套完整的軟體發展程序，以便讓專案工程

師開發產品。發展程序所產生的軟體應具高度模組化，且包括在各個階段中產生完整文件資料，以供未來新產品開發時的參考。

(2) 為了瞭解整個產品線包括該產品的特徵，要有能描述不同產品特性的模型，以區分產品間的共同性與差異性。

(3) 考慮未來客戶可能的需求改變，即產品可能會面臨的變化。

在本論文中，我們將提出一套以開發有效開發初版產品為前提的產品線軟體發展方法。我們所提出的方法是以 RUP (Rational Unified Process)[16][17]為基礎，並結合特徵模型的優點，使之適用於產品線的發展。RUP 是一項以使用案例為驅策的軟體發展程序，而特徵適用來表達不同版本產品的共同性與差異性，因此我們藉由提供特徵模型與適用案例的聯繫關係，讓產品線的發展者以特徵模型描述新產品的特色，藉由特徵模型與使用案例的聯繫關係，找到各特徵的相對應適用案例，而後依序 RUP 的完整程序完成軟體的發展過程。另外，在產品線的架構設計上，我們亦根據常見的版本變化，提出適當的設計準則。

以下，我們將先在 2 與 3 小節中分別介紹 RUP 與特徵模型，而後在 4 與 5 小節中再討論特徵模型與適用案例的關係，並在 6 與 7 小節中討論產品線的架構設計，綜合前面所提出的觀念，我們提出的程序稱為 Unified Feature-Oriented Product Line (UFOPL)發展程序，我們在 8 小節中簡介該程序。最後，我們討論本方法的缺點與未來工作。

2. RUP 簡介

RUP 是強調系統是藉由不斷反覆(iterative)的開發，並且以架構為中心、使用案例為驅策的一套軟體發展程序。整個程序主要分為四個階段，分別是開始(Inception)、製作(Elaboration)、建構(Construction)、轉移(Transition)。同時每一階段分別執行主要的工作流程，分別是業務模型(Business Modeling)、需求(Requirements)、分析與設計(Analysis & Design)、實作(Implementation)、測試(Test)與部署(Deployment)(如圖 2-1 所示)，以及組構管理(Configuration Management)等支援流程(Supporting Process)。完整的軟體發展週期就再歷經這四個階段，每個階段又區分為若干個循環(Iteration)，在每一個循環經過各工作流程後，得到一階段性的版本呈現給顧客並最為下一個階段開始的依據(強調系統開發重複的特性)。

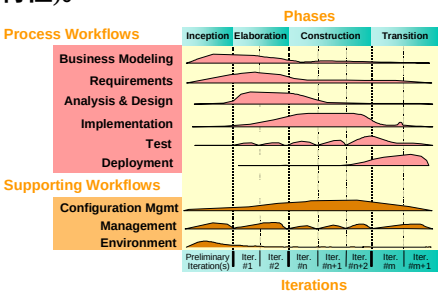

圖 2-1 RUP 示意圖

在 RUP 中，主要的產物包括有：

- 使用案例模型(Use case model)：使用案例模型包括所有的使用案例、角色，還有彼此間的關係。
- 分析模型(Analysis model)：分析模型主要工作除了將使用案例做更詳細的改進，同時將系統的行為做初步的分配，定義每一個物件所負責的責任。
- 設計模型(Design model)：設計模型定義了系統的靜態結構部分，如系統底下的子系統、類別與介面間的關係。另外還有動態部分，包括將使用案例所描述的動作實現的合作圖表，描述子系統、類別和介面間的互動。
- 部署模型(Deployment model)：部署模型定義實際的節點與系統之間的關係，及描述每一個節點所負責的任務。
- 實作模型(Implementation model)：實作模型包括實際可執行的程式碼部分，這些元件是依據設計階段設計的類別等產物，實際產生出來的的元件、程式碼。
- 測試模型(Test model)：測試模型描述測試個案，用來驗證實際產物是否符合使用案例的描述。

上述的各項模型 RUP 均用 4+1[[18]的觀點來描述(如圖 2-2 所示)，所謂 4 是指是邏輯觀點、實作觀點、程序觀點、部署觀點，1 是指使用案例觀點。各項觀點分述如下：

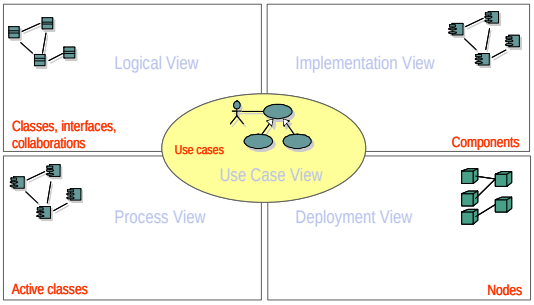

圖 2-2 4+1 模型架構圖[18]

3. 特徵與特徵模型

特徵與特徵模型的觀念事由 K. C. Kang 等人在其著名的領域分析方法 FODA[14]與 FORM[13]。他們在其方法[1]提出一套有系統的系統分析程序，重點在於捕捉同一領域下不同應用間的共同性及差異性，利用特徵對系統作分析後的產物，發展出適用於領域下的架構及元件，以做為可再利用的開發成果。其分析步驟有過程中幾個重要產物，第一個會得到也是最重要的是特徵模型 (Feature Model)，特徵模型對於不管是在發展可再利用的領域製品(Domain Artifacts)，或在應用產品的發展過程中使用領域製品(Domain Artifact)，都有具有極大的便利性。

特徵模型的建立及特徵的定義在 FORM 中極為重要，因為在系統開發的過程中，顧客跟工程師之間試圖藉由溝通來達成對系統的認知。但顧客表達的方式跟工程師所接收到的資訊可能會有出入。而負責開發軟體的工程師之間會有不同的角色問題，分析師跟

[1] FORM 是 Kang 延伸 FODA 的精神所提出的方法，FODA 中著重的是在需求工程階段分析的結果，而 FORM 則擴充到軟體的設計及實行上，規定如何由特徵模型發展領域架構及可再利用元件。在本小節中，我們以 FORM 為基礎介紹特徵與其發展方法。

工程師所強調的角度可能又不一樣。所以定義了「特徵」這樣一個共同的名稱，來當作顧客跟工程師之間對系統溝通的媒介。而一般在需求層次並不容易從再利用的角度，分析可再利用元件單位的探討，所以建構一個特徵模型除了能對系統輪廓有整體性的瞭解外，還可以探討可再利用的空間。

FORM 將「特徵」定義為顧客跟工程師之間，彼此溝通想法的基礎。藉由特徵的描述，顧客跟工程師可以用共同的語言來描述整個系統。顧客用特徵表達他們所要的功能，而工程師也可以用特徵來作為彼此間開發系統的共同專業術語。所以對顧客來說，產品的特徵就是列出一連串顧客希望產品為他們做的事；而對工程師來說，特徵則是偏向他們所預期系統能提供的功能、及其輪廓架構。

以 FORM 的定義，主要可以將特徵分成四種類別：

(1) **能力特徵 (Capability feature)**：能力特徵將一個產品可能有的系統服務、系統運作的方式、提供給使用者的功能、效能等，盡可能將有關產品輪廓的呈現表示出來。同時能力特徵還可以進一步細分為功能性特徵跟非功能性特徵，且功能性又可再細分為系統服務、系統動作。

(2) **作業環境特徵 (Operating environment feature)**：產品在使用操作的環境因素，也是影響發展系統的主要特徵之一。像與產品作業有關的硬體設備（每一個跟系統有關係的節點）、相關作業系統環境（如 Windows98）、資料庫擷取方式等，都是屬於產品在作業環境方面的特徵。

(3) **領域技術特徵 (Domain technology)**：領域技術特徵指的是低層執行系統的技術，與實作技巧特徵的差別在於，領域技術特徵著重在產品相關領域下獨有的執行技術，在此領域下領域技術並不適用在其他領域。如在航空領域下的航海術，就不能運用在知識管理領域的執行技術上。

(4) **實作技巧特徵 (Implementation technique)**：實作技巧特徵指的是更低層執行系統的技術，與領域技術特徵的差別在於，實作技巧特徵專指比領域技術特徵更一般化的執行技術，在其他不同領域下也會有的技巧。如演算法中使用排序的方法是 queue 或 stack、使用的編輯器是 vi 或 joe 等，在別的領域下也會用到的實作技巧皆屬此範疇。

除了這四種主要的分類架構外，FORM 還會依據每一個特徵所顯示的資訊，再將每一層的特徵細分為不同的範疇。所以 FORM 提出的過程中，最重要的一個步驟就是建立特徵模型。在一個領域裡以特徵為基礎建立的特徵模型，是為了幫助工程師訂定一個標準，藉由定義標準的專業術語以方便溝通問題。同時可用來作為同一領域下不同應用程式的評估標準，從特徵所呈現出領域的特性，也可以作為跟其他不同領域的比較依據。而特徵模型的建立可以分成下面三個主要的步驟：

(1) **特徵確認**：特徵的確認是根據領域專家還有其他文件資料如使用者手冊、設計文件等分析後的結果。領域專家泛指對領域熟悉的角色，如系統使用者、領域分析師、領域開發者等。藉由領域專家對領域的瞭解，從不同角色的角度確認他們所注重的特徵。所以建立特徵模型的第一步，就是要先將所有的特徵確認出來。

(2) **特徵分類**：當確認完所有的特徵之後，就要將這些特徵分類。這些特徵會依他們所呈現出來的資訊加以分類，主要的分類結果可以分成四種類別：能力特徵、作業環境特徵、領域技術特徵、實作技巧特徵。

(3) **特徵組織與分析**：當特徵分類成四層時，接下來就要把這四層的特徵組織起來，建立特徵間彼此的關係。通常一個特徵都不會是獨立的，他一定會跟其他特徵有合作的關係，或是可以在細分為更小的特徵。如一個服務特徵可能會在細分成更小的服務特徵，或一個能力特徵可能必須要伴隨在特定的作業環境下，或某種領域特徵只能選用幾種特定的實作技巧方法等。通常能力特徵、領域技術特徵、實作技巧特徵彼此間會有伴隨的關係，而作業環境特徵則是關係整個領域作業環境的重要因素。

在確認完 FORM 對特徵的分類方式，接下來就可以討論當特徵改變時，受到影響的物件類型更動情況。可以知道 FORM 主要將特徵分成的四層分類架構，從最上層的能力特徵、受能力特徵決策而影響的的領域技術特徵/實作技巧特徵 到涵蓋軟硬體方面的作業環境特徵共四層。每一層特徵還可再依特徵所呈現出的資訊，再更細分為幾層物件類型[15]，如圖 3-1 所示：

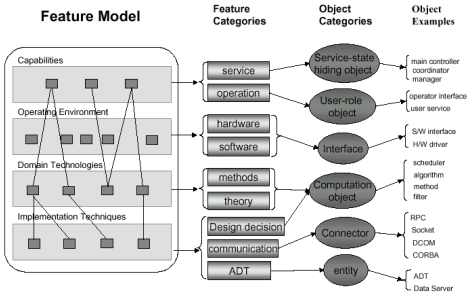

圖 3-1 特徵類別與物件類別關係圖

4. 特徵與使用案例

如同我們在前面所提及的，特徵已經被公認是最適合描述產品線的方法。所謂特徵泛指產品的特色、獨特的性質，用來描述同一領域的產品，以區分和其他成員不同的地方。一般來講，特徵涵蓋的範圍包括有產品使用者的特徵、系統的整體結構、所提供的服務項目等都是屬於特徵的一部份。從使用者的角度來看產品，不同的產品會有不同角度的功能性特徵。從系統的角度，各個產品包含使用不同的系統服務、建構在不同的環境下、使用與支援不同的外部設備，這些亦是特徵的來源。所以利用特徵來區分產品，涵蓋各個系統功能性需求(Functional Requirement)，非功能性需求(Non-functional Requirement)以及相關設計與實作上的考量。

特徵也是產品所呈現出來的特性，用已區分在同一個產品線上不同家族成員的區分。特徵分成好幾種類型，會根據在系統發展過程中角色的不同而有所區分：

- 從產品使用者的角度來看，特徵會是系統可以提供的功能、動作部分，如服務特徵等。

- 從產品分析師的角度來看，特徵是他們注重的領域技術。如航空領域的航海術、銀行資金轉帳方式、飛彈導航系統的技術，還有圖書館資訊管理、書籍分類方法、搜尋方式等。
- 從產品開發工程師的角度來看，特徵是實際實作的技術。如實作程式碼時所使用的資料結構、演算法等。

從產品線的角度來看，特徵可以提供不同的角度來來分辨與其他產品的不同，例如，系統提供的功能性服務，還有非功能性如發展作業平台的差異、效能、外部硬體支援等，都是用以區分不同產品的特徵。為了因應市場需求的快速變化，發展者必須能夠快速的因應市場上的需求，針對現有的產品線進行修改，因此，如何精確辨認新產品的特徵，並依此進行產品的開發與相關產品線的再利用就變得很重要。

然而，在 RUP 是以使用案例做為產品開發的主導，使用案例是代表系統與系統外的作用者(Actor)之間為了某種目的所產生的互動行為。

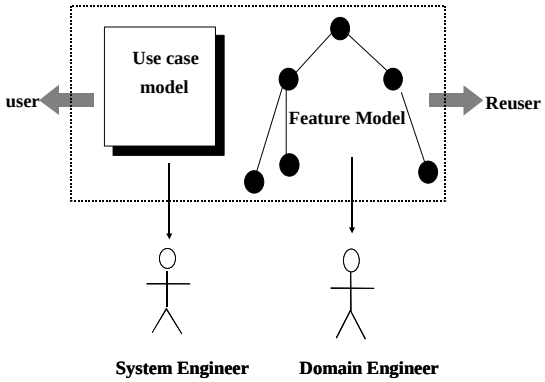

圖 4-1 使用案例與特徵[11]

如圖 4-1 所示，使用案例是系統工程師用來描述單一系統使用者在不同情況下對系統的需求，而特徵則是領域工程師從需求與設計中萃取軟體的特色以便再利用。從其描述的內容來看，使用案例與特徵共同的部分是在涵蓋功能性需求，雖然使用案例著重在使用者與系統互動關係的描述，從功能性需求的角度對各種使用情況作一詳盡的描述，而能力特徵僅止於羅列所描述的領域的功能，但是兩者均是在描述功能性的需求，只是描述的細緻程度與目的不同，我們可以說功能性的能力特徵是將系統的使用案例作一分類與歸納，以便再利用者可以方便的找到所需要的功能性需求，否則對於一複雜的系統來說，直接從使用案例的角度來進行再利用常常會因為使用案例的個數與描述過於繁瑣，而難以分辨出所需要的特徵。至於非功能性的需求，RUP 中是以文字性的敘述方式參雜在需求規格中(或是使用案例模型中的備忘錄 – Note)，而特徵模型則是條列在能力特徵上，兩者在細緻程度上則無太的不同。

除了功能性與非功能性的需求外，特徵包含在設計時的考量與作業環境(作業環境通常是非功能性需求的一部份，但不見得會特別明白表示)，這兩個部分對再利用者來看，再利用者應當對於所需要的系統需求應當有相當的瞭解，對於設計上的考量也有基本的認識，所以如果能夠提供這兩方面的訊息，再利用者應當可以更有效的找到所需要的再利用資源。

5. 建立特徵與使用案例的聯繫關係

在 FODA 的想法裡，特徵分析是為了達成與使用者協議而建立在領域下一般化的功能模組。這樣的結果很像是使用案例模型所要表示的，主要是因為當 FODA 在提出這樣想法的時候，使用案例尚未在物件導向需求分析的軟體工程領域中被廣泛接受，而現在已經可以很清楚的確定特徵分析是為了預先考慮到不同情況下需求捕捉的技術。特徵模型與使用案例模型的建立有不同角度的目的，特徵模型是以再利用者為導向；而使用案例模型則是以使用者為導向。使用案例模組化減輕了在特徵模組化過程中所會面臨的無法清楚說明的負擔，同時使用案例收集並詳細描述使用者對系統功能性的需求；而特徵模型則集中在如何從領域工程將領域中的共同性與差異性有系統化地組織，以探討可再利用的部分。目前仍然存在需要我們去克服的問題在於，要如何建立一個完整的特徵模型與使用案例間關係圖。

通常系統服務只靠一兩個使用案例的描述是不夠的，由於使用案例主要描述系統功能性服務部分，所以由數個使用案例所包成的使用案例包裹，會滿足系統所要提供給使用者的服務或動作。而在特徵模型中的能力特徵涵蓋系統功能性部分，所以在能力特徵中細分的服務或動作，所對應到使用案例模型中的是每一個使用案例包裹。而能力特徵又會關係到底層實作的方法，同樣的使用案例模型也會影響到分析設計階段的結果。所以藉由特徵與使用案例包裹的關係，可以整合特徵模型與發展過程產生軟體設計文件間的關係。

從特徵模型的角度來看，特徵會依呈現的資訊分成四種不同的類型，*能力特徵*可再分為服務特徵與動作特徵，涵蓋系統功能性需求。所以每一個能力特徵所對應的是以一到多個使用案例的連結關係，也就是以每個使用案例包裹(use case package)為單位可以對應到能力特徵的服務或動作。*作業環境特徵*主要涵蓋非功能性需求，可在細分為硬體環境與軟體環境部分。作業環境的特徵會反應在需求階段的補充文件上(supplementary)，紀錄與系統有關的環境變因。所以該類特徵變化在分析階段功能性需求的部分不受影響，主要在設計階段系統設計時會影響。*領域技術特徵*的類型可再細分為方法、理論與演算法等，主要為了滿足能力特徵提供功能性需求的條件。所以該類特徵提供解決問題的方法，對應到軟體產物中分析階段的解決方法，在分析類別中每一個類別的責任都有可能受到影響。同時分析結果的改變會影響系統設計，在設計的類別中定義的方法也是受到分析結果改變的主要部分。*實作技巧特徵*主要是為了實現領域技術不同的實作技巧，屬於底層的執行方法，包括有溝通方法和資料抽象化等方法。該類特徵會依實作時程式設計師使用的程式語言與實際作業環境的不同而有所差異，會影響設計階段的系統設計結果。所以該特徵的對應的軟體設計文件主要屬於以服務子系統為單位的設計結果，同時也會對應到分析階段分析包括的結果。

因此，如果要結合特徵模型在產品線的好處與 RUP 完整發展方法的好處，我們必須提供使用案例與特徵模型的聯繫關係，讓系統工程師所定義的不同產品的使用者案例以及相關的軟體產物，領域工程師

經由領域分析將產品線的產品共通與差異點以特徵模型。另外，完整的產品線的描述除了包含特徵模型外，也應包含使用案例等軟體產物，對特徵模型不足的部分做更進一步的描述。特徵模型與 RUP 的軟體產物的關係圖如圖 5-1 所示。能力特徵所提供的服務與動作，會對應到需求階段的使用案例包裹，該使用案例包裹內的每個使用案例代表該能力特徵的不同應用情境。藉由使用案例包裹與服務包裹本身的追蹤的關係，每一個服務特徵都會以服務包裹為單位實現。在特徵模型中為了滿足一個服務特徵，會有一個以上的領域技術特徵來實現，領域技術特徵所提供的是解決問題的方法，所以對應到軟體產物中分析階段的解決方法。而每一個領域技術又會有不同實作的技術，對應到軟體產物中的是設計階段實際實作的方法，會跟實作時所用的程式語言或作業環境因素有關。所以為了滿足系統功能性的需求，能力特徵與領域技術特徵、實作技巧特徵間有互相組合的關係，而其與軟體產物的對應關係分析，都要藉由使用案例的對應來建立。

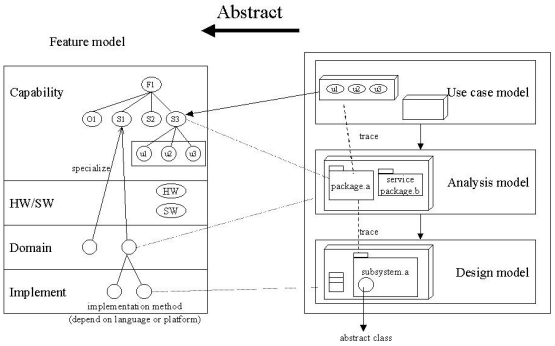

圖 5-1 特徵模型與 RUP 軟體產物關係圖

6. 產品線產品變化的問題分析

所謂產品線是指一群特徵相關的產品，彼此有共通性，也有差異性，前面所談的特徵模型，目的在於利用依系統化的方式來表示這些共通性與差異性，但從產品線的角度來看，軟體架構的設計必須能夠包含這兩者，而產品線發展程序也必須將這可能的情況納入考慮。因此，在本小節中，我們將討論，產品線差異性的來源。目前產品線上產品變化的問題主要有四種情況，

(1) **功能性變化**：功能性變化是產品線中各個不同產品的主要來源，隨著產品本身演進的過程，產品也會隨著版本的增加，擴充或修改上一個版本的功能，Microsoft Word2000 是針對架構在 97 的功能上擴充一些新的功能，這應當是產品線最容易發生的狀況。另外，為了區隔不同市場，產品線通常會區分為若干個版本，每個版本皆是針對不同的市場對象，如以學生為目標的教育版，一般使用者的普及版，以及專業使用者的專業版，還有為了吸引新使用者的是用版等，每個版本皆在功能上有所差異，例如教育版只有基本的功能，而專業版則擁有所有甚至若干附加的功能，而適用版則是專業版但加上使用上的限制。

(2) **作業平台變化**：作業平台變化也是產品差異化的主要來源，作業平台的變化主要是考量作業系統以及包含底層的硬體，目前商用或專屬的作業系統眾多，如果在特定的領域(如電信，控制系統等)則作業系統的種類經更加繁多，通常為了市場的考量，同一產品會支援相近的作業系統，如支援 Palm 的產品通常也會支援 WinCE，支援 Linux 的也支援 UNIX。因此，在產品線的開發時，時常必須將一產品移植到其他的作業系統，以擴大產品縣的市場規模。

(3) **國際語言變化**：國際語言變化主要是針對不同國家(以及不同語言)使用者的需求，一般而言，國際語言的變化僅止於將軟體上使用者會看到的文字(如程式的使用者介面的所出現的文字，使用者手冊，說明等)，但也有情況是在不同的語言變化中加入功能變化的考量。此一部份多半肇因於對某一語言的市場上的考量，為了重視該市場而加入或修改某些功能以符合當地的使用文化。另一種情況則是因為作業系統本身在不同的國際語言上的差異，所以產品線也必須作適當的調整，但這可歸類於作業環境的變化。

(4) **程式語言改版**：發生程式語言特徵改變的情況，主要有可能是因為非功能性的需求改變。為了滿足需求的變化，針對系統底下的模組進行修改，不變更到系統整體大架構。所以會發生主要原因為需求改變的原因有以下幾種情況。(1)非功能性需求的改變：如系統效能提升的需求，為了增加某子系統的運作效率，經評估後將該子系統改用新的程式語言開發。(2) 介面物件重新設計：如為了增加使用者介面的親和力，將介面獨立重建。像用 Fortune 開發的系統，因為 Fortune 本身缺乏介面功能，為了增加介面的使用性，將介面部分選用 Visual Basic 呈現。程式語言改版的影響，主要在設計階段時設計一個類別時會有所變化。

系統功能改版的變化屬於能力特徵的改變，在特徵模型中屬最上層的服務或動作特徵改變。能力特徵表示系統所能提供的功能性服務，所以其改變會影響跟使用案例集合為單位的使用案例包裹。作業平台改變的變化顯示產品存在的作業環境條件，改版的影響反應在特徵模型作業環境特徵上，屬於非功能性特徵，在產品需求文件的改變，會記錄在補充文件資料裡。作業環境特徵關係產品發展時設計階段的考量，所以作業平台改版的變化要到設計階段考慮產品架構設計時才考慮。程式語言改版的情況，多數反應非功能性需求的改變上。針對產品非功能性的需求，為改善系統效能或功能不夠的非功能性需求，會在特徵模型的作業環境特徵上顯現出來，以補充文件的方式記錄新產品特徵的改變，在分析設計階段再針對其影響改變設計結果。國際語言改版是為了作業環境條件改變考慮，配合不同國家有不同語言的問題，所以針對不同語言編碼的語系不同，國際語言改版的問題反應在作業環境條件的改變，也是以補充文件的方式記錄不同國際語言版本特徵的改變，再考慮設計上的問題與軟體架構的影響。

所以產品線上會面臨的問題，除了系統功能改版的問題外，大都反應在作業環境的條件上。而特徵模型間四層不同類型特徵間的關係，會隨著上層如能力特徵的改變，影響底層領域技術特徵或實作技巧特

徵，因為提供同樣的服務特徵有不同的實作方法，而且會為了滿足作業環境的條件或配合其他特徵的限制而不同。

上述的這幾個狀況對產品線而言，除了以特徵模型來區分其變化的狀況外，更重要的就是軟體架構的設計，軟體架構描述軟體內部所有組成元件的結構與元件之間的關係，以作為後續軟體發展的藍圖。因為軟體架構是軟體發展中最核心的一部份，好的產品線軟體架構才能有好的產品線，產品線的軟體架構必須考慮這些可能的變化，讓這些變化的情況可以很容易的融入，而成為產品的架構。所以在下一小節中將會分析每一種版本改變的情況，提出進行產品線軟體開發前的架構設計原則。

7. 產品線軟體架構設計原則

由於不同版本的改變對系統設計影響部分都不一樣，若考慮在產品線上發展產品時，若要在最短的時間內改版，就要利用前一版的產品設計架構。所以在談產品線軟體發展前最重要的工作，就是要做好在產品線上系統架構的設計前提，有了這樣的前提就可以針對版本改變時對系統架構與設計上的影響。不同版本改變對軟體架構與設計的影響討論如下。

(1) **功能性變化**：功能性的變化是軟體發展中最常遇到的情況，但其變化也最多，即使過去非產品線的情況下，功能性變化仍舊是設計上考量的重點，因此許多設計的原則，如模組化(Modulization)、抽象化(Abstraction)、階層化(Layering)、物件化與元件化(Component)皆有助於提升軟體架構的彈性以應付功能性的變化。另外，特徵與使用案例的追蹤關係以及使用案例與其他軟體產物的關係，也有助於架構的調整。

(2) **作業平台變化**：作業平台改變對軟體架構的影響，主要出現在作業平台所提供的系統服務上的改變，設計階段設計軟體架構時。我們可將系統架構可分成應用程式層、中介軟體層、系統軟體層。中介軟體層位於應用程式與系統軟體之間的軟體，負責提供介面的服務工作，讓應用程式可以藉由中介軟體提供的服務作業在預設的軟體作業環境上。所以作業平台的改變關係著不同層次間系統架構的關係，主要的改變會是在介面物件的設計上。而介面的設計是為了處理作業平台改變對應用程式造成的影響。所以從作業平台改變對系統架構的影響來看，應用程式要能夠滿足原來的需求的話，就要由介面來幫助處理應用程式下層環境。

作業平台的改變到設計階段才產生影響。考慮應用程式與底層作業系統間的架構，應用程式要求作業系統提供服務的方式有兩種：一種是直接透過介面物件呼叫系統呼叫，由系統呼叫中斷作業系統要求系統服務；另一種則是透過介面物件與中介軟體溝通，再由中介軟體處理下層與系統呼叫間的動作。所以在架構設計上要考慮的主要是介面物件設計的問題，處理與作業系統溝通的方式，以及考慮當作業系統改變時不同系統程式改變的情況。

(3) **程式語言變化**：程式語言改變的影響在設計階段，主要的改變在於在設計一個類別。考慮設計類別本身的動作、參數、屬性、類別，這些都會因程式語言的改變而有語法上的改變。而設計類別中的方法(method)提供實現設計類別中動作的方法，通常以自然語言或 pseudo code 的形式當作是實作的註解，所以程式語言的改變並不會影響方法，受影響的是類別實作的結果。而設計類別間的關係表示被其他類別包含的語意象徵，不受程式語言改變的影響。至於設計子系統的架構關係不受影響，設計子系統包含設計類別，其改變的部分在設計類別改變時即考慮到。

(4) **國際語言變化**：會使用不同語系的作業平台版本。從軟體架構來看，我們假設應用程式層以下包括中介軟體與底層作業系統就有支援多國語系的情況下考慮發展不同國際語言版。由於不同國際語言版本可能會有功能上的差異，比如中文版 word 就比英文版 word 在排版上多了多了直書/橫書的格式功能，而考慮改版的前提下是系統要提供相同的功能服務，新增或修改的功能會在系統功能改版的部分考量。所以需求的改變會影響設計階段的結果，尤其反應在介面的呈現上，因為不同語系有不同的編碼方式，所以在呈現的格式與設計上要重新考慮。

國際語言改變的影響主要在軟體架構上使用者介面設計上的改變，因為不同語系的編碼方式不同，如英文編碼單位 1byte 而中文則是 2bytes，所以在介面的呈現上要考慮因語系改變而造成介面設計上，與編碼有關的呈現部分要獨立設計。如功能表單、對話視窗、圖表說明文字等。另外還有欄位的長度與位置，會因語系間不同長度的編碼方式而改變，位置也會隨之調整。除了介面設計上的改變化，資料儲存格式也會受影響，所以在分析階段要確認有哪些儲存資料會因為語系的改變而受影響，然後在設計階段才將這些類別獨立設計，考慮資料儲存的資料型態改變。

8. UFOPL 發展程序

UFOPL 的發展程序正是結合以特徵模型為主的領域分析技術與 RUP 的軟體發展程序。UFOPL 主要分兩個階段。第一階段為領域工程(Domain Engineering)階段，主要工作在於建立一個產品線的參考模型。參考模型的主要目的在於描述產品線上，經由 UFOPL 發展程序後得到的軟體產物與特徵模型間的關係，稱為 UFOPL_1。第二階段主要工作為應用工程(Application Engineering)階段，在於利用第一階段得到的結果參考模型，在產品線上當版本變化時，如何再利用參考模型，產生新產品，稱為 UFOPL_2。而 UFOPL_1 和 UFOPL_2 關係圖如圖 4-1 所示。

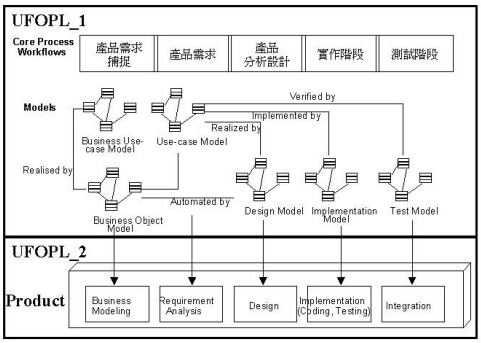

圖 8-1 UFOPL_1 and UFOPL_2 **關係圖**

8.1. UFOPL_1

UFOPL **第一階段主要工作，即領域工程的階段，目的在在於建立產品線的參考模型，包括描述產品線的特徵模型與其他相關軟體產物。**UFOPL_1 **主要流程如下：**

(1) **產品線需求捕捉：在此階段，領域工程師經由對產品線的瞭解，相關的產品需求，以及其他類似可供參考的領域模型，捕捉產品線的需求，以及定義該產線的範圍。在此階段的步驟遵循** RUP **中需求捕捉的** 4 **個步驟。只是原本是系統分析師扮演此一角色，而現在由領域工程師執行此工作，而考慮的需求也擴大為產品線的需求。**

(2) **產品線需求： 產品線分析的主要目的在於針對產品線的需求，特別在前一階段已經指明為使用案例的功能性需求作更進一步的分析，以便得到更明確的使用者與系統的互動關係。**

(3) **產品線分析：產品線分析是針對產品線的使用案例作更進一步的分析，找到各使用案例的的分析模型，包含分析類別**(Analysis Class)**、使用案例具體化關係**(Use Case Realization)**與 分析包裹**(Analysis Package)**組合而成。**

(4) **產品線設計階段：在這階段主要產物威為設計模型，主要由設計類別**(Design Class)**、使用案例具體化關係與設計子系統**(Design Subsystem)**彼此間的關係組合而成。其中描述特殊需求的需求部分，以文件描述的方式收集在同一份補充文件**(supplementary)**裡，這些非功能性需求會在設計階段由作為設計師設計系統架構的依據。**

(5) **產品線實作階段：在這階段的主要產物為實作模型，主要由可執行元件、檔案等組合而成，其他還包括實作子系統及彼此間的關係等。**

(6) 產品線測試階段：在這階段主要產物為測試模型，測試模型描述系統應該如何被測試。測 試階段中的測試案例主要來源可分兩種。一種是來自需求階段的使用案例，做的是黑箱測試，根據使用案例描述的需求訂定測試案例，測試系統是否符合需求描述。另一個來源來自設計階段的 use case realization-design**，做的是白箱測試，根據設計階段設計系統的考量訂定測試案例，測試設計結果的架構是否有問題。所以當產品線上特徵改變時，會受到影響的測試案例主要從與特徵相關的使用案例追蹤相關測試案例的黑箱測試部分；還有在設計部分對設計結果或系統架構的修改，也會影響到測試案例的白箱測試部分。**

8.2. UFOPL_2

在 UFOPL **第一階段時，可以得到如圖** 4-2 **所示參考模型，描述特徵模型與做完** UFOPL **第一階段後的程序產物互動關係。**

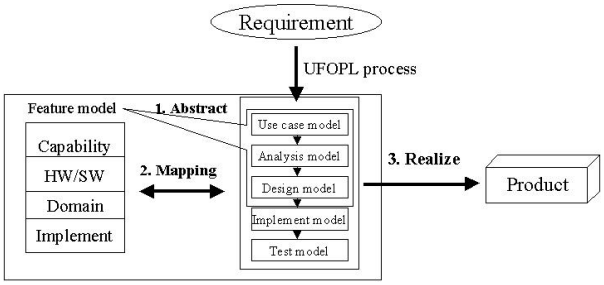

圖 8-2 **參考模型圖**

由上圖可知，發展人員在需求階段就開始考慮產品線的特徵變化，當進行 UFOPL **的發展程序時，分析師在需求階段開始抽出，一直到分析、設計階段的結果，都可以藉由最原始的使用案例，找出特徵相對應的產物。所以利用** UFOPL **第一階段產生的參考模型，在第二階段進行版本改變的修改，找出因需求改變而受影響的部分，再討論不同版本改變的修改，其流程如圖** 8-3 **所示。**

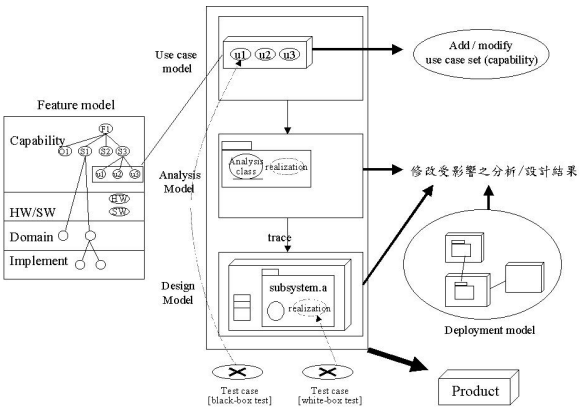

圖 8-3 UFOPL-2 **流程圖**

9. 結論

在傳統物件導向軟體工程研究中以物件導向為基礎的方法中，探討開發過程中可再利用的資產，一直是軟體工程研究努力的方向。為了解決以往各個產品獨立作業的問題，且讓所有工程師遵循一個定義明確的軟體程序，以期更有效提升軟體發展速度，並減少發展過程中所需的成本與花費。本論文提出以產品線為對象，提供完整的產品發展程序及方法。

同公司開發同一系列的產品，通常具有某種程度的同質性，而享有共同特徵及共同商業目標的軟體集合，這些產品間的關係可構成一系列的產品線。所以我們考慮一般軟體公司開發同一產品線同系列產品會遇到的可能，因應不同需求的改變，在產品線上會面臨的版本變化主要包括有四種情況：

(1) 國際語言變化改版

(2) 作業平台改版

(3) 系統功能改版

(4) 程式語言改版

由於一個成功的再利用策略是將軟體再利用對整個發展過程提升的效益，擴充到整個產品線上的產品開發速度。我們希望將再利用策略對整個產品發展過程提升的效益，擴充到以公司為單位的產品線上。目前相關研究在產品線上提出的方法中主要的問題在於，不是沒有特徵的想法就是產品線的定義不夠清楚，同時也沒有一套完整且有系統的程序結合到正統軟體發展方法上。所以為了解決這些問題，目前研究中以 Kang 提出的 FORM[13]與 Martin 提出的 FeatuRSEB[11]最完善，但仍有發展程序不夠完整，或特徵與相對應架構界定模糊的問題。所以本論文對這些缺點進行改進，提出能夠結合特徵與完整軟體發展程序的產品線式軟體發展程序。

為了解決目前相關研究面的問題，我們提出以特徵為基礎的產品線式軟體發展程序。針對目前產品線發展方法的研究上最重要的缺點上，也就是，缺乏完整的程序，我們參考目前最廣為使用的 RUP 發展程序，將之與特徵模型結合，提出以特徵模型與 RUP 為基礎的以特徵為導向之統一產品線式(UFOPL)發展程序想法，建立特徵模型與軟體發展程序中各項產物間的關係，並且依照前述四種產品線不同的情況，討論相對應的發展程序。

UFOPL 的發展程序是結合以特徵模型為主的領域分析技術與 RUP 的軟體發展程序。UFOPL 主要分兩個階段。

(1) 第一階段主要工作為領域工程階段，在於建立一個產品線的參考模型，包括描述產品線的特徵模型與其他相關軟體產物。參考模型的主要目的在於描述產品線上，經由 UFOPL 發展程序後得到的軟體產物與特徵模型間的關係，稱為 UFOPL_1。

(2) 第二階段主要工作為應用工程階段，在於利用第一階段得到的結果參考模型，在產品線上當版本變化時，如何再利用參考模型，產生新產品，稱為 UFOPL_2。

(3) 藉由這套程序，產品線的領域工程師可以系統化的方式發展產品線的參考模型，而且產品工程師也可以依據此一包含完整軟體產物的參考模型，快速的發展新的產品。

未來我們希望針對所提出的方法進行相關的實驗，並且提出適當的 UFOPL 輔助工具，希望能將發展過程階段產物，以一套有系統的管理方式存進資料庫，並試圖研究資料庫知識分類機制，結合知識管理資料庫的相關研究。

10. 相關文獻

[1] Charles W. Krueger, "Software Reuse," *ACM Computing Surveys*, Vol. 24, No. 2, pp. 131-183, June 1992.

[2] Even-André Karlsson, *Software Reuse: A Holistic Approach*. NY: John Wiley & Sons, 1995.

[3] Ivar Jacobson, et. Al., *Software Reuse: Architecture Process and Organization for Business Success*, New York: ACM Press; 1997.

[4] M. L. Griss, "Software Reuse: From Library to Factory," *IBM Sys. J.*, Vol. 32, No. 4, pp. 548-566, 1993.

[5] W. Frakes and S. Isoda, "Success Factors of Systematic Reuse," *IEEE Software*, pp. 15-19, Sept. 1994.

[6] D. M. Weiss, and C.T.R. Lai, *Software Product-Line Engineering: A Family-Based Software Development Process*. NY: Addison-Wesley, 1999.

[7] J. Bosch, *Design & Use of Software Architectures – Adopting and Evolving a Product-Line Approach*. NY: Addison Wesley, 2000.

[8] Peter Knauber and Giancarlo Succi (editors), *Proc. of Software Product Lines: Economics, Architectures, and Implications, Limerick, Ireland, June 10th 2000*

[9] P. Donohoe (editor), *Software Product Lines Experience and Research Directions*. Boston: Kluwer Academic Publishers, August, 2000

[10] Martin Griss, Implementing Product-Line Features with Component Reuse, *Proc. 6th Intl. Conf. Software Reuse*, Vienna, Austria, Jun 2000

[11] Martin Griss, et. Al., "Integrating Feature Modeling with the RSEB," *Proc. of Intl. Conf. on Software Reuse*, Victoria, Canada. June 1998.

[12] Bayer, J., Flege, et. Al., "PuLSE: A methodology to develop software product lines," *Proc. Software Reusability (SSR'99)*, May 1999.

[13] Kyo C. Kang, et. Al., "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures," *Annals of Software Engineering*, Vol. 5, pp. 143-168, 1998.

[14] Kyo C. Kang, et. Al., *Feature-Oriented Domain Analysis (FODA) Feasibility Study, Software Engineering Institute*. CMU/SEI-90-TR-021, February 1990.

[15] Kwanwoo Lee, et. Al., "Feature-Based Approach to Object-Oriented Engineering Applications for Reuse," to be published in *Software: Practice and Experience*, 2000.

[16] Philippe Kruchten, *The Rational Unified Process: An Introduction*, 2nd edition, Addison Wesley, 2000

[17] Ivar Jacobson, Grady Booch, and James Rumbaugh, *The Unified Software Development Process,* Reading, MA: Addison-Wesley,1999

[18] P.B. Kruchten, The 4+1 View Model of Architecture, *IEEE Software*, Vol. 12, No. 6, Nov. 1995, pp.42-50